UNIVERSIDADE FEDERAL DO PIAUÍ
CENTRO DE CIÊNCIAS DA NATUREZA - CCN
DEPARTAMENTO DE COMPUTAÇÃO – DC
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

RELATÓRIO
Symbol Table

- Análise da implementação de uma tabela de simbolos binária e sequencial, verificando a eficiência de sua inserção e busca.

PAULO EDUARDO RAMOS DE ARAUJO
Teresina – PI
2019

A implementação do programa foi feita com o uso das classes disponibilizadas no site https://algs4.cs.princeton.edu , as classes usadas foram :

- BinarySearchST.java
- SequentialSearchST.java
- Queue.java
- StdIn.java
- StdOut.java
- Stopwatch.java

O programa instancia duas tabelas de simbolos, uma sequencial e outra binária, essas tabelas são preenchidas ao longo do programa e são feitas verificações do tempo de execução gasto para cara metodo.

É utilizado uma variável **n** que guarda o tamanho do preenchimento inicial das tabelas, e a cada ciclo de teste o valor de **n** é dobrado. Para a contagem dos ciclos é utilizado a variavel **k** que é incrementada no final do teste. O valor adotado para **n** foi 1000.

A cada ciclo é feito o preenchimento da tabela de simbolos com um laço de repetição, até ela atingir o tamanho especificado em **n**, a key e o value passado são o ciclo do laço e um número aleatório (gerado pela classe Random da biblioteca *Java.util.Random)* menor que **n,** respectivamente. Após o preenchimento é feito a busca de 50 chaves existentes na tabela e 50 chaves não existentes na tabela. Para garantir a existencia de uma chave é criado um número aleatório menor que **n,** e para garantir a não existência é somado **n** ao número gerado.

```
for (int i = binaryST.size(); i < n; i++) {
      binaryST.put(i, gerador.nextInt(n));
}

for (int i = 0; i < 100; i++) {
      if (i < 50) { // 50 buscas com sucesso.
            binaryST.get(gerador.nextInt(n));
      } else { // 50 buscas falhas.
            binaryST.get(n + gerador.nextInt(n));
      }
                  }
```

Código de inserção e busca da ST binária.

Durante os testes iniciais foi percebido que a inserção da tabela de simbolos sequencial tem seu tempo de execução muito alto, então a tabela de simbolos sequencial só é testada para valores de **n** menor que 500.000.

A execução do arquivo do programa gera quatro gráficos que mostram o desempenho dos metodos de inserção e remoção para cada tipo de tabela de simbolos.

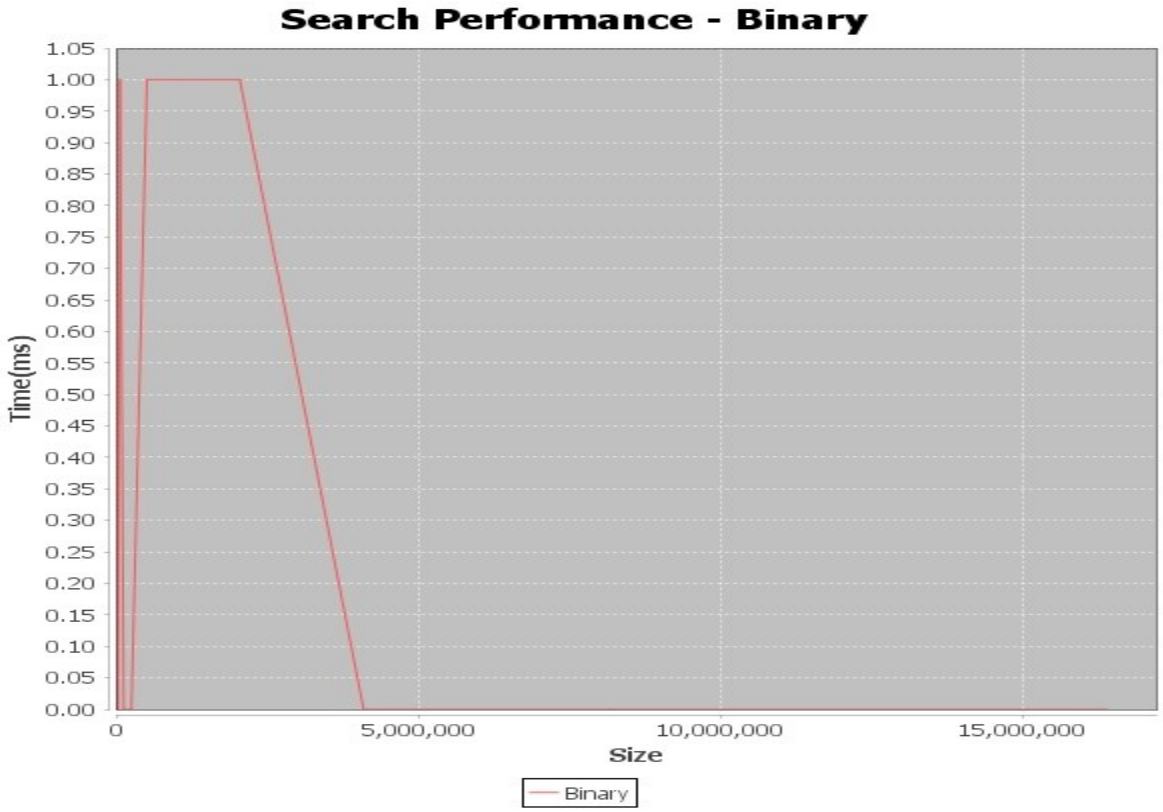


Figura 1 – Performance da busca na tabela de simbolos binária.

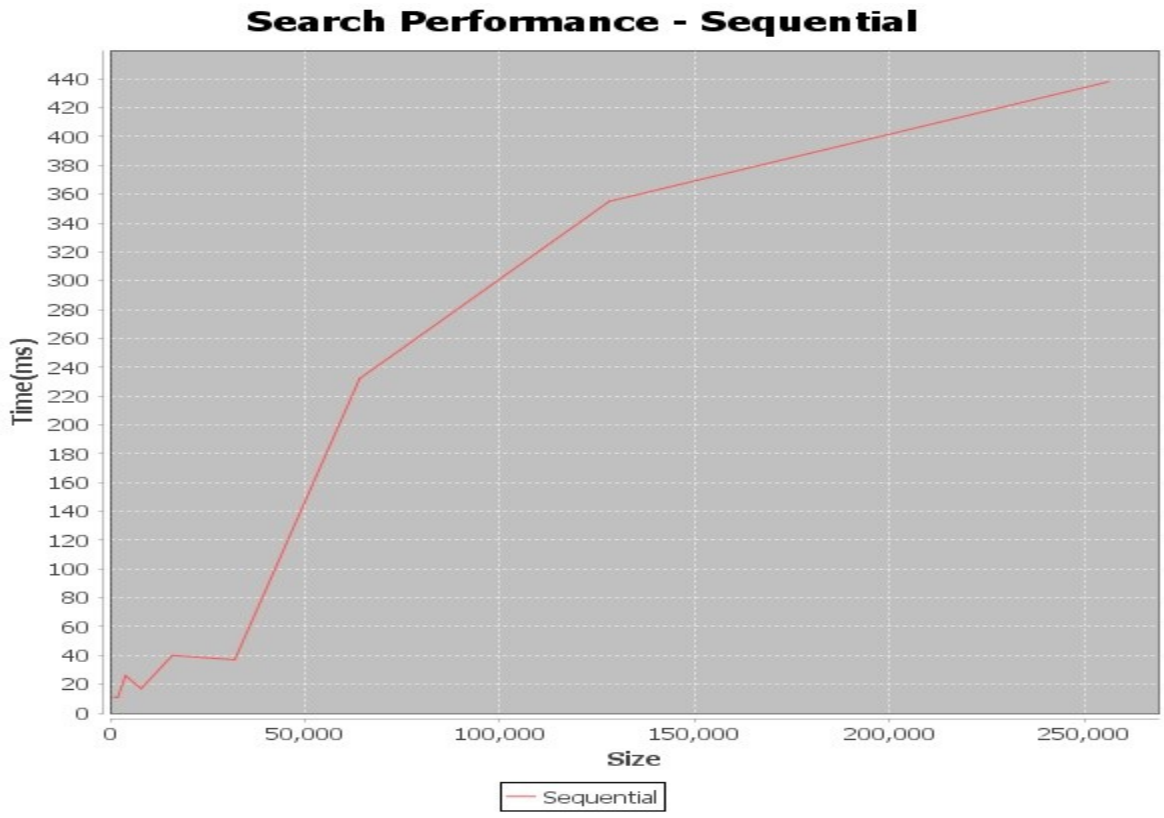


Figura 2 – Performance da busca na tabela de simbolos sequencial.

Como pode ser observado na figura 1, a busca binária se mostra bastante eficiente, ela mantem o limiar de 1 ms para busca em tabelas com até 15 milhões de simbolos. Já a busca sequencial tem seu tempo de execução crescendo de forma exponencial para tabelas com poucos simbolos, tendo uma demora de até 440 ms para tabelas com 250mil simbolos, como podemos observar na figura 2.

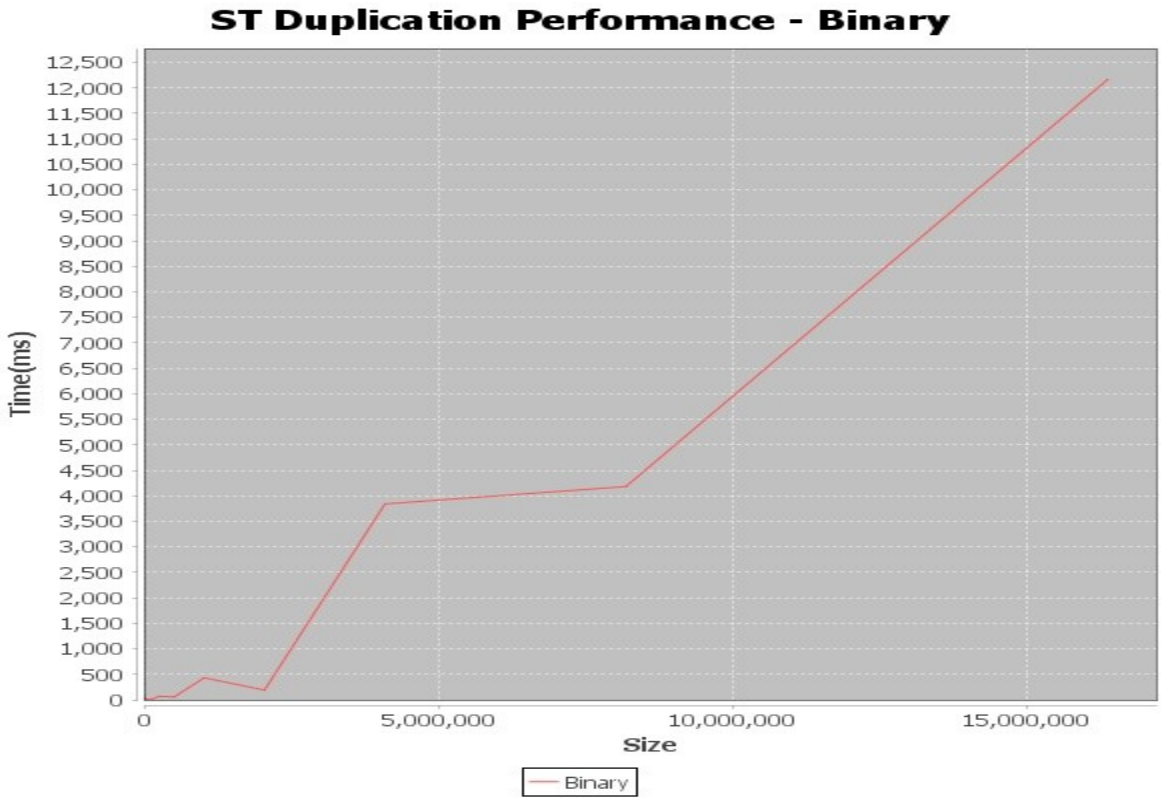
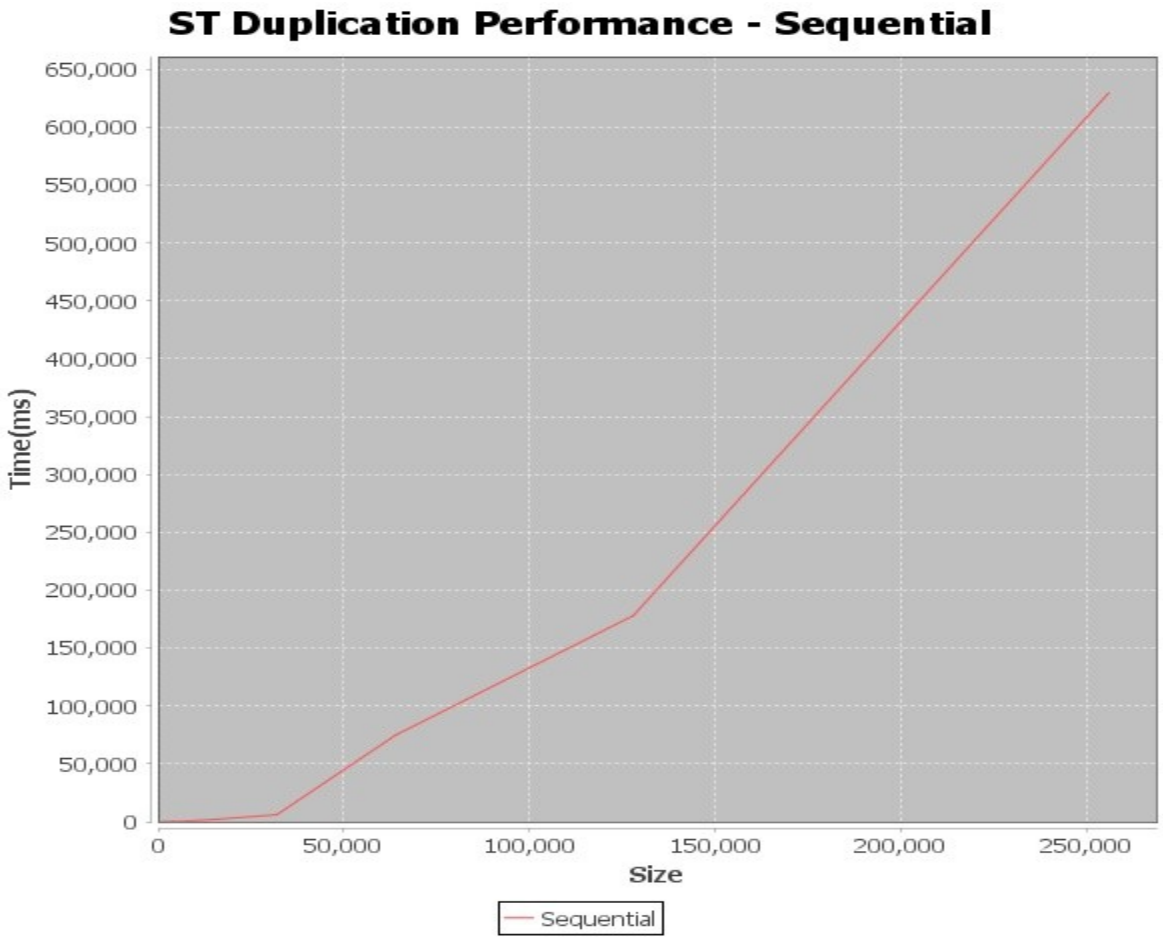


Figura 3 – Performance de duplicação da tabela de simbolos binária.

A figura 3 mostra a tabela a perfomance da tabela binária durante a inserção de elementos. A tabela de simbolos binária tem seu tempo de inserção chegando a uma média de 1,5 milhões de simbolos por segundo, mostrando-se bastante eficaz para tabelas com tamanhos exorbitantes.

Figura 4 – Perfomance de duplicação da tabela de simbolos sequencial.



A tabela de simbolos sequencial tem seu tempo de inserção bastante elevado, atingindo até um minuto para inserir 30 mil simbolos, como exemplifica a figura 4.

CÓDIGO:

```
	public static void main(String[] args) throws IOException {
		Integer n = 1000;
		Integer k = 1;
		long aux_time;
		Stopwatch contador;
		BinarySearchST<Integer, Integer> binaryST = new BinarySearchST<>();
		SequentialSearchST<Integer, Integer> sequentialST = new
SequentialSearchST<>();
		Random gerador = new Random();
		do {
		// Testando tabela binária
			contador = new Stopwatch();
			for (int i = binaryST.size(); i < n; i++) {
				binaryST.put(i, gerador.nextInt(n));
			}
			aux_time = contador.elapsedTime();
			addBinary.add((double) n, aux_time);

			contador = new Stopwatch();
			for (int i = 0; i < 100; i++) {
				if (i < 50) { // 50 buscas com sucesso.
					binaryST.get(gerador.nextInt(n));
				} else { // 50 buscas falhas.
					binaryST.get(n + gerador.nextInt(n));
				}
			}
			aux_time = contador.elapsedTime();
			binary.add((double) n, aux_time);
		// Testando tabela sequencial
			if (n < 500000) {
				contador = new Stopwatch();
				for (int i = sequentialST.size(); i < n; i++) {
					sequentialST.put(i, gerador.nextInt(n));
				}
				aux_time = contador.elapsedTime();
				addSequential.add((double) n, aux_time);
				contador = new Stopwatch();
				for (int i = 0; i < 100; i++) {
					if (i < 50) { // 50 buscas com sucesso.
						sequentialST.get(gerador.nextInt(n));
					} else { // 50 buscas falhas.
						sequentialST.get(n + gerador.nextInt(n));
					}
				}
				aux_time = contador.elapsedTime();
				sequential.add((double) n, aux_time);
			}

			k++;
			n = 2 * n;

		} while (k <= 15);
	}
```

CONCLUSÃO

Foi implementado dois tipos de Tabela de Simbolos, binária e sequencial, em seguida as duas arvores foram submetidas a testes de desempenho para adicionar elementos e buscar elementos.

A tabela de simbolos sequencial mostrou-se menos eficiente que a tabela binária, seu tempo de inserção e busca é bastante elevado e mostrou-se ineficaz para tabelas maiores que 250 mil. Já a tabela binária mostrou um limiar de 1 ms para busca em tabelas de até 15 milhõs de simbolos, e sua inserção mostrou um ganho de desempenho enorme em relação a inserção da tabela sequencial.